# Random Sum-Product Networks: A Simple but Effective Approach to Probabilistic Deep Learning

**Robert Peharz**[1], **Antonio Vergari**[2], **Karl Stelzner**[3], **Alejandro Molina**[3], **Xiaoting Shao**[3], **Martin Trapp**[4]
**Kristian Kersting**[3], **Zoubin Ghahramani**[1,5]
University of Cambridge[1], MPI for Intelligent Systems[2], TU Darmstadt[3], TU Graz[4], Uber AI Labs[5]
{rp587,zoubin}@cam.ac.uk, antonio.vergari@tue.mpg.de,
{stelzner, molina,xiaoting.shao, kersting}@cs.tu-darmstadt.de, martin.trapp@tugraz.at

## Abstract

Sum-product networks (SPNs) are a particularly promising type of deep probabilistic model that allows an exceptionally rich set of exact and efficient inference scenarios. To achieve this, though, SPN have to obey specific structural constraints. While SPN structure learning received much attention, most of the proposed methods so far are tedious to tune, typically do not scale easily and hinder integration with deep learning frameworks. In this paper, we investigate how important structure learning in SPNs actually is. To this end, we propose a "classical deep learning approach", i.e., generate an unspecialized random structure scaling up to millions of parameters, and then apply modern GPU-based optimizers with regularization. That is, we investigating the performance of SPNs in the absence of carefully selected structures. As it turns out, our models perform on par with state-of-the-art SPN structure learners and deep neural networks on a diverse range of generative and discriminative scenarios. Most importantly, they yield well-calibrated uncertainties, thus standing out among most deep generative and discriminative models in being robust to missing features and detecting anomalies.

## 1 INTRODUCTION

An intelligent system should both be able to deal with uncertain inputs, as well as express its uncertainty over outputs. Especially the latter is a crucial point in automatic decision-making processes, such as medical diagnosis and planning systems for autonomous agents. Therefore, it is no surprise that probabilistic approaches have recently gained great momentum in deep learning, which has led to a variety of probabilistic models such as variational autoencoders (VAEs) [49, 30], generative adversarial nets (GANs) [26], neural auto-regressive density estimators (ARDEs) [32, 55, 56], and normalizing flows (NFs) [20, 31].

However, most of these probabilistic deep learning systems have limited capabilities when it comes to *inference*. First, they have to resort to approximate inference in most (e.g., marginalization and conditioning for ARDEs, NFs) – if not all (e.g., VAEs) – inference scenarios, while sometimes not providing even access to an explicit likelihood function (e.g. GANs). Furthermore, even when tractable approximations can be carried out, there is no guarantee that these computations yield a calibrated estimation of the underlying uncertainty in data, or even conform to human expectations [11, 39].

In this landscape, *sum-product networks* (SPNs) [13, 45] are a promising avenue, as they are a class of deep probabilistic models permitting *exact* and *efficient* inference. In particular, SPNs are able to compute *any* marginalization and conditioning query in time linear of the model's representation size. This property is a hallmark of SPNs, distinguishing them from the other probabilistic models mentioned above. Nevertheless, despite their attractive inference properties, SPNs have received comparatively limited attention in the deep learning community. A major reason for this is that the structure of an SPN needs to obey certain constraints, to deliver the promised tractable inference. This requires either to carefully design the structure by hand or to learn it from data [15, 23, 41, 50, 42, 57, 2, 16, 54, 46, 17, 38].

These structural requirements of SPNs are opposed to the usual homogeneous structures employed in deep learning. Therefore, they hinder a flawless integration into deep learning frameworks. Additionally, learning SPN structures has proven hard to scale, precluding them from being used on e.g. large scale image tasks.

In this paper, *we investigate how important structure learning in SPNs actually is*. Specifically, we seek to establish what performance can be achieved with SPNs whose structure has not been carefully tuned to the dataset at hand. To this end, we introduce a simple and scalable method to construct *random and tensorized SPNs* (RAT-SPNs), waiving the necessity for structure learning: we first construct a random *region graph* [15, 41], which we subsequently populate with arrays of SPN nodes. This strategy essentially dictates a random hierarchical tensorial decomposition [52], leading to SPNs with reduced sparsity. RAT-SPNs map well onto deep learning frameworks like Tensorflow [1], and allows to scale to millions of parameters by automatically taking advantage of GPU-parallelization.

For density estimation, i.e. the generative case, we use the classical expectation-maximization (EM) algorithm [14], which has recently been derived for SPNs [43]. Since EM is free of tuning-parameters and rapidly increases the likelihood, it is a natural choice for this task. We show that this simple strategy yields test-likelihoods surprisingly close to ID-SPN [50], one of the most sophisticated SPN learners available.

In addition, we show that RAT-SPNs, when trained discriminatively, yield classifiers competitive to deep neural nets. So far, no principled discriminative SPN structure learner is available while discriminative parameter learning has been mainly applied to images – relying either on powerful feature extraction [22] or specialized structures [3, 52, 48]. Our discriminative RAT-SPNs are domain-agnostic and thus applicable in a much wider setting.

Most importantly, *we demonstrate that RAT-SPNs deliver well-calibrated uncertainties*: they can be used to reliably detect anomalies and are robust under missing data. In contrast to deep classifiers, hybrid discrete-generative RAT-SPNs can explicitly quantify when they are not confident about their predictions. Furthermore, generative RAT-SPNs are not fooled by certain out-of-domain image detection tests on which VAEs, NFs, and ARDEs consistently fail [11, 39].

The start off by reviewing the required background and discussing related work. Subsequently, we introduce RAT-SPNs and our proposed tensorized learning schemes. Then, we thoroughly evaluate RAT-SPNs empirically w.r.t. the current SPN learning as well as deep neural learning for generative and discriminative modeling. Finally, we conclude and present future work.

## 2 BACKGROUND & RELATED WORK

We denote random variables (RVs) by upper-case letters, e.g. $X$, $Y$, and their values by corresponding lower-case letters, e.g., $x$, $y$. Similarly, we denote sets of RVs by upper-case bold letters, e.g., $\mathbf{X}$, $\mathbf{Y}$ and their combined values by corresponding lower-case letters, e.g., $\mathbf{x}$, $\mathbf{y}$.

An SPN $\mathcal{S}$ over $\mathbf{X}$ is a probabilistic model defined via a directed acyclic graph (DAG) containing three types of nodes: *input distributions*, *sums* and *products*. All leaves of the SPN are input distribution functions over some subset $\mathbf{Y} \subseteq \mathbf{X}$. Inner nodes are either weighted sums or products, denoted by $\mathsf{S}$ and $\mathsf{P}$, respectively, i.e. $\mathsf{S} = \sum_{\mathsf{N} \in \mathbf{ch}(\mathsf{S})} w_{\mathsf{S},\mathsf{N}} \mathsf{N}$ and $\mathsf{P} = \prod_{\mathsf{N} \in \mathbf{ch}(\mathsf{P})} \mathsf{N}$, where $\mathbf{ch}(\cdot)$ denotes the children of a node. The sum weights $w_{\mathsf{S},\mathsf{N}}$ are assumed to be non-negative and normalized: $w_{\mathsf{S},\mathsf{N}} \geq 0$, $\sum_{\mathsf{N}} w_{\mathsf{S},\mathsf{N}} = 1$.

The *scope* of an input distribution $\mathsf{N}$ is defined as the set of RVs $\mathbf{Y}$ for which $\mathsf{N}$ is a distribution function, i.e. $\mathbf{sc}(\mathsf{N}) := \mathbf{Y}$. The scope of an inner (sum or product) node $\mathsf{N}$ is recursively defined as $\mathbf{sc}(\mathsf{N}) = \bigcup_{\mathsf{N}' \in \mathbf{ch}(\mathsf{N})} \mathbf{sc}(\mathsf{N}')$. To allow for efficient inference, SPNs should satisfy two structural constraints [13, 45], namely *completeness* and *decomposability*. An SPN is complete if for each sum $\mathsf{S}$ it holds that $\mathbf{sc}(\mathsf{N}') = \mathbf{sc}(\mathsf{N}'')$, for all $\mathsf{N}', \mathsf{N}'' \in \mathbf{ch}(\mathsf{S})$. An SPN is decomposable if it holds for each product $\mathsf{P}$ that $\mathbf{sc}(\mathsf{N}') \cap \mathbf{sc}(\mathsf{N}'') = \emptyset$, for all $\mathsf{N}' \neq \mathsf{N}'' \in \mathbf{ch}(\mathsf{P})$. In that way, all nodes in an SPN recursively define a distribution over their respective scopes: the leaves are distributions by definition, sum nodes are mixtures of their child distributions, and products are factorized distributions, assuming (conditional) independence among the scopes of their children.

Besides *representing* probability distributions, the crucial advantage of SPNs is that they permit *efficient inference*: In particular, any marginalization task reduces to the corresponding marginalizations at the leaves (each leaf marginalizing only over its scope), and evaluating the internal nodes as usual in a bottom-up pass [44]. Conditioning is tackled in a similar manner. It is important to note that these inference scenarios are rendered tractable by the above mentioned structural constraints, which are a critical aspect when learning SPNs.

Indeed, how to build an SPN structure is a central topic in many works in the literature, starting from [45]. There, an SPN structure tailored to images was proposed, which recursively partitions images using horizontal and vertical splits, and models the distributions of sub-images using a collection of sum nodes. Dennis and Ventura [15] improved this architecture by using non axis-aligned splits, found by k-means applied to the transposed data matrix. Peharz et al. [41] introduced a bottom-up approach to learn SPN structures, using an information-bottleneck method. Gens and Domingos [23] proposed a general high-level scheme called *LearnSPN* which follows a hierarchical co-clustering ap-

proach, i.e. it alternately clusters data instances – corresponding to sum nodes – and splits variables – corresponding to product nodes – by independence testing. Since then, there have been several improvements of the basic LearnSPN scheme, such as regularization by employing multivariate leaves [57], employing an efficient SVD-approach [2], generating compacter networks by merging tree-structures into general DAGs [46], learning product nodes via multi-view clustering over variables [28] or lowering their complexity by approximate independence testing [18], and learning SPN structures over hybrid domains [38]. Rooshenas and Lowd [50] refined LearnSPN by learning leaf distributions using Markov networks represented by arithmetic circuits [36]. The resulting SPN learner, called ID-SPN, still delivers state-of-the-art results for density estimation, at least when considering single models (ensembles can improve results [34, 19]). In [52] a hierarchically structural mixture model was proposed, which can be understood as an SPN with convolutional structure tailored to image data.

While structure learning is indisputably a relevant topic in SPNs, the "antithesis" has received surprisingly little attention: *How important is detailed structure learning in SPNs actually? Akin to deep neural networks, can we get decent models by just scaling up a random SPN structure and applying simple parameter estimation techinques?* The current successes deep learning methods arguably make this approach intriguing. In addition to that, structure learning is rather tedious in SPNs. For example, correctly implementing and running ID-SPN is non-trivial, not least because ID-SPN has a "huge parameter space" as Roosheans and Lowd stress [50].

Thus, it is fair to say that the special structural requirements of SPNs have hindered their wider use in practice. In particular, interesting combinations with other deep learning models have so far remained relatively unexplored. Thus, random SPNs, as introduced in this paper, are a promising direction for probabilistic deep learning.

# 3 RANDOM SUM-PRODUCT NETWORKS

In order to construct our *random and tensorized SPNs* (RAT-SPNs), we use the notion of a *region graph* [15, 41] as an abstract representation of the network structure.

Given a set of RVs $\mathbf{X}$, a *region* $\mathbf{R}$ is defined as a non-empty subset of $\mathbf{X}$. Given any region $\mathbf{R}$, a $K$*-partition* $\mathcal{P}$ of $\mathbf{R}$ is a collection of $K$ non-overlapping sub-regions $\mathbf{R}_1, \dots, \mathbf{R}_K$, whose union is again $\mathbf{R}$, i.e. $\mathcal{P} = \{\mathbf{R}_1, \dots, \mathbf{R}_K\}$, $\forall k\colon \mathbf{R}_k \neq \emptyset$, $\forall k \neq l\colon \mathbf{R}_k \cap \mathbf{R}_l = \emptyset$, $\bigcup_k \mathbf{R}_k = \mathbf{R}$. In this paper, we consider only 2-partitions, which causes all product nodes in our SPNs to have exactly two children. This assumption, frequently made in the SPN literature, simplifies SPN design and seems not to impair performance.

**Algorithm 1** Random Region Graph

**procedure** RANDOMREGIONGRAPH($\mathbf{X}, D, R$)
  Create an empty region graph $\mathcal{R}$
  Insert $\mathbf{X}$ in $\mathcal{R}$
  **for** $r = 1 \dots R$ **do**
    SPLIT($\mathcal{R}, \mathbf{X}, D$)

**procedure** SPLIT($\mathcal{R}, \mathbf{R}, D$)
  Draw balanced partition $\mathcal{P} = \{\mathbf{R}_1, \mathbf{R}_2\}$ of $\mathbf{R}$
  Insert $\mathbf{R}_1, \mathbf{R}_2$ in $\mathcal{R}$
  Insert $\mathcal{P}$ in $\mathcal{R}$
  **if** $D > 1$ **then**
    **if** $|\mathbf{R}_1| > 1$ **then** SPLIT($\mathcal{R}, \mathbf{R}_1, D-1$)
    **if** $|\mathbf{R}_2| > 1$ **then** SPLIT($\mathcal{R}, \mathbf{R}_2, D-1$)

A *region graph* $\mathcal{R}$ over $\mathbf{X}$ is a connected DAG whose nodes are regions and partitions such that i) there is exactly one region $\mathbf{R} = \mathbf{X}$ without parents (i.e. $\mathbf{X}$ is the *root region*), ii) all leaves of $\mathcal{R}$ are regions, iii) all children of regions are partitions and all children of partitions are regions (i.e. $\mathcal{R}$ is bipartite), iv) if $\mathcal{P}$ is a child of $\mathbf{R}$, then $\bigcup_{\mathbf{R}' \in \mathcal{P}} \mathbf{R}' = \mathbf{R}$ and v) if $\mathbf{R}$ is a child of $\mathcal{P}$, then $\mathbf{R} \in \mathcal{P}$.

Given a region graph, we can easily construct a corresponding SPN as follows: Populate each leaf-region with a collection of $I$ input distributions, and all other regions with a collection of sum nodes. For the root region we create $C$ sum nodes, and for all internal regions, we create $S$ sum nodes. Finally, for all partitions, take all cross-products of nodes contained in the child-regions, and connect these products as children of all sums in the parent region. Pseudo-code for this procedure is provided in the supplementary.

We denote the $C$ sum nodes in the root region as $\mathcal{S}_c(\mathbf{X})$, $c = 1, \dots, C$. For density estimation, we assume $C = 1$, in which case the single root readily represents a correctly normalized density $\mathcal{S}(\mathbf{X}) := \mathcal{S}_1(\mathbf{X})$. For classification, the $C > 1$ roots represent *class-conditional* distributions $\mathcal{S}_c(\mathbf{X}) =: \mathcal{S}(\mathbf{X} \,|\, Y = y)$, $y \in \{1, \dots, C\}$. A sample $\mathbf{x}$ is classified by applying Bayes' rule: $\mathcal{S}(Y \,|\, \mathbf{x}) = \frac{\mathcal{S}(\mathbf{x} \,|\, Y)\, P(Y)}{\mathcal{S}(\mathbf{x})} = \frac{\mathcal{S}(\mathbf{x} \,|\, Y)\, P(Y)}{\sum_y \mathcal{S}(\mathbf{x} \,|\, y)\, P(y)}$. The class-prior $P(Y)$ can be estimated from the empirical class-distribution, or just be fixed to, e.g., uniform.

We construct random regions graphs – and thus RAT-SPNs – with the simple procedure depicted in Algorithm 3: We randomly divide the root region into two sub-regions of equal size (possibly breaking ties) and proceed recursively until depth $D$, resulting in an SPN

of depth $2D$. This recursive splitting mechanism is repeated $R$ times. An example of a RAT-SPN is illustrated in the supplementary.

It is easy to verify that the number of sum-weights in RAT-SPNs is given as $W_{\mathsf{S}} =$

$$\begin{cases} RCI^2 & \text{if } D = 1, \\ R\left(CS^2 + (2^{D-1} - 2)S^3 + 2^{D-1}SI^2\right) & \text{if } D > 1. \end{cases} \quad (1)$$

Similarly, we can count the parameters of the input distributions, which we assume to factorize into univariate distributions. In this case, it follows that the total number of parameters for the input distributions is

$$W_{\mathsf{D}} = RI|\mathbf{X}|P, \quad (2)$$

where $P$ is the number of parameters per univariate distribution.

We implemented Alg. 3 in Python and the corresponding RAT-SPNs in Tensorflow.[1] The input distributions are Gaussians for real data and categorical for discrete data. All computation are performed in the log-domain to avoid numerical underflow. Sum-weights, which are required to be non-negative and normalized, are reparameterized via log-softmax layers. To perform summations in the log-domain, we use the *log-sum-exp* trick.

We have just introduced how to build a complete and decomposable SPN structure in a randomized manner, which is able to comprise millions of parameters while being easily mapped to GPU-accelerated tensor computations. Next, we consider learning RAT-SPNs both in a *generative* and *discriminative* setting.

### 3.1 GENERATIVE LEARNING

For generative learning, we assume that we have a training set $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ of i.i.d. samples drawn from an unknown distribution $P^*(\mathbf{X})$, which we wish to approximate. The canonical approach to generative learning is maximizing the log-likelihood

$$\mathsf{LL}(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N} \log \mathcal{S}(\mathbf{x}_n), \quad (3)$$

where $\boldsymbol{w}$ denotes all parameters of the SPN, i.e. sum-weights and parameters of the input distributions. Note that by construction, $\mathcal{S}(\mathbf{X})$ is already a correctly normalized distribution over $\mathbf{X}$.

To optimize (3), we use the standard Expectation-Maximization (EM) algorithm [14], which has been recently derived for SPNs in [43].[2] EM rapidly and monotonically increases the likelihood, is free of tuning-parameters and can be implemented via simple forward and backward evaluations to compute the required expected sufficient statistics – see [43] for details. These very convenient properties fit well the philosophy to adopt the "simplest and most effective" learning strategy for RAT-SPNs.

### 3.2 DISCRIMINATIVE LEARNING

For discriminative learning, we focus on classification. Let $\mathcal{X} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ be a training set of inputs $\mathbf{x}_n$ and class labels $y_n$. We train RAT-SPN classifiers by minimizing the *cross-entropy*

$$\mathsf{CE}(\boldsymbol{w}) = -\frac{1}{N}\sum_{n=1}^{N} \log \frac{\mathcal{S}_{y_n}(\mathbf{x}_n)}{\sum_{y'} \mathcal{S}_{y'}(\mathbf{x}_n)}, \quad (4)$$

which is equivalent to maximizing the *conditional log-likelihood* $\sum_n \log \mathcal{S}(y_n \mid \mathbf{x}_n)$, when assuming a uniform class prior. Furthermore, we can readily combine (3) and (4) into a *hybrid generative-discriminative* [6] loss

$$\mathsf{H}(\boldsymbol{w}) = \lambda\, \mathsf{CE}(\boldsymbol{w}) - (1 - \lambda)\, \frac{\mathsf{LL}(\boldsymbol{w})}{|\mathbf{X}|}, \quad (5)$$

which trades off cross-entropy and log-likelihood. For $\lambda = 1$, we retrieve pure discriminative learning, while for $\lambda = 0$, we retrieve pure generative learning. For $0 < \lambda < 1$, we are allowing our RAT-SPN classifiers to also capture the distribution over $\mathbf{X}$, a crucial feature to deal with uncertainty over inputs, e.g., in presence of missing values. The likelihood $\mathsf{LL}$ is obtained by marginalizing the class variable $Y$, as illustrated above. For discriminative learning, we use Adam with default hyper-parameters and a fixed batchsize of 100.

### 3.3 PROBABILISTIC DROPOUT

The size of RAT-SPNs can be easily controlled via the structural parameters $D$, $R$, $S$ and $I$. As usual in deep learning, we want RAT-SPNs structures to be overparameterized. However, as they are likely to overfit, we need to introduce regularization. To this end, we perform *early stopping* by monitoring the loss on a validation set. Furthermore, we propose two variants of the *dropout* heuristic [53] for RAT-SPNs: at inputs and at sum nodes.

*Dropout at inputs* essentially marks input features as missing at random. Following the probabilistic

[1] We will publicly release all the code to reproduce the work upon acceptance.

[2] The original derivation in [45] contained an error. Moreover, note that the concave-convex procedure later on proposed in [59] coincides with EM updates for sum-weights, but is, in general, distinct for input distributions.

paradigm, we simply wish to marginalize over these missing features. Fortunately, this is an easy task in SPNs, as we only need to set the input distributions corresponding to a dropped-out features to value 1. A similar criterion was used in a convolutional variant of SPNs [52], which drops out small image patches, however.

We introduce *dropout at sum nodes*, by setting their child-products to 0 (in fact $-\infty$ in log-domain) with a certain probability. This introduces additional states for the latent variables associated to the mixtures represented by sum nodes [43], effectively eliminating a random subset of mixture components by setting them to 0.

# 4 EXPERIMENTS

Our aim is to empirically evaluate RAT-SPNs on a wide range of tasks and real world benchmarks. First, we investigated how accurate they are as density estimators in the generative setting, comparing them to several state-of-the-art SPN learners. Second, we empirically checked if they are on par with deep neural networks in the discriminative setting, over a diverse set of classification domains. Finally, we measured how well-calibrated uncertainties measured by RAT-SPNs are by employing them for anomaly detection and classification under missing inputs, two scenarios on which current deep architectures fall short [39, 11].

## 4.1 GENERATIVE LEARNING: RAT-SPNs ARE COMPARABLE TO STATE-OF-THE-ART

For the generative setting, we evaluated RAT-SPNs on 20 standard benchmark datasets used to compare SPN learners [23]. We remark that the objective of this experiment is *not* to yield new state-of-the-art log-likelihoods on these datasets. Instead, we aim to understand the impact of structure learning compared to simple parameter optimization of random structures by investigating how close RAT-SPNs can come to more sophisticated SPN learning schemes.

To this end, we directly compared RAT-SPNs to the prototypical SPN structure learner, LearnSPN [23]; to the extension proposed in [18], LearnSPN-RGVS, which approximates the statistical tests to introduce product nodes in a random fashion; and to OBMM [47] where a LearnSPN-like structures generated in a random fashion is fed into Bayesian parameter learning[3]. By doing so, we evaluate learning RAT-SPNs versus full structure learning (LearnSPN), a randomly-flavored variant (LearnSPN-RGVS), and random structure with sophisticated parameter learning (RandSPN+OBMM). Additionally, we report the current state-of-the-art log-likelihood on the 20 datasets as achieved by ID-SPN [50] for structure learning and CCCP for parameter learning when used to fine-tune post-trained structures obtained with LearnSPN.

Table 1: Average test log-likelihoods for the 20 benchmark datasets for generative learning. Best results for each dataset are in bold, statistically significant results are denoted by ∘. As one can see, RAT-SPNs are competitive to LearnSPN, LearnSPN-RGVS, and RandSPN+OBMM (see text), while scoring log-likelihoods that are en-par with SPN state-of-the-art parameter (LearnSPN+CCCP) and structure learning (ID-SPN) routines.

| dataset | LearnSPN | LearnSPN-RGVS | RandSPN+OBMM | RAT-SPN | LearnSPN+CCCP | ID-SPN |
|---|---|---|---|---|---|---|
| nltcs | -6.110 | -6.370 | -6.070 | ∘**-6.011** | -6.029 | ∘-6.020 |
| msnbc | -6.113 | -6.113 | **-6.030** | ∘-6.039 | -6.045 | ∘-6.040 |
| kdd-2k | ∘-2.182 | - | -2.140 | ∘**-2.128** | -2.134 | ∘-2.134 |
| plants | **-12.977** | 16.784 | -15.140 | -13.439 | -12.872 | -12.537 |
| jester | -53.480 | 54.968 | -53.860 | ∘**-52.970** | -52.880 | ∘-52.858 |
| audio | ∘-40.503 | 41.935 | -40.700 | ∘**-39.958** | -40.020 | ∘-39.794 |
| netflix | -57.328 | 59.842 | -57.990 | **-56.850** | -56.782 | -56.355 |
| accid. | **-30.038** | 40.232 | -42.660 | -35.487 | -27.700 | -26.983 |
| retail | ∘-11.043 | 11.336 | -11.420 | ∘**-10.911** | -10.919 | ∘-10.847 |
| pumsb. | **-24.781** | 42.423 | -45.270 | -32.530 | -24.229 | -22.405 |
| dna | **-82.523** | 99.266 | -99.610 | -97.232 | -84.921 | -81.211 |
| kosarek | ∘-10.989 | 11.490 | -11.220 | ∘**-10.888** | -10.880 | ∘-10.599 |
| msweb | -10.252 | -11.001 | -11.330 | **-10.116** | -9.970 | -9.726 |
| book | ∘-35.886 | 35.665 | -35.550 | ∘**-34.684** | -35.009 | ∘-34.137 |
| e.movie | ∘**-52.485** | 64.456 | -59.500 | -53.632 | ∘-52.557 | ∘-51.512 |
| web-kb | ∘-158.204 | 167.547 | -165.570 | ∘**-157.530** | -157.492 | ∘-151.838 |
| reut.52 | ∘**-85.067** | 97.271 | -108.010 | ∘-87.367 | -84.628 | ∘-83.346 |
| 20ng | -155.925 | - | -158.010 | **-152.062** | -153.205 | -151.468 |
| bbc | ∘**-250.687** | 269.029 | -275.430 | ∘-252.138 | -248.602 | ∘-248.929 |
| ad | ∘**-19.733** | 57.545 | -63.810 | -48.472 | -27.202 | ∘-19.053 |

We cross-validated the split-depth $D \in \{1, 2, 3, 4\}$ and the number of sum-weights $W_{\mathsf{S}} \in \{10^3, 10^4, 10^5\}$. In order to yield a particular $W_{\mathsf{S}}$, we used (6) to select values for $R$, $S$ and $I$. These values were picked a-priori such that they were roughly balanced and approximately yield a targeted $W_{\mathsf{S}}$ (see supplementary). We used soft EM for 100 epochs and used early stopping for regularization. No dropout was applied in the generative case.

Average test log-likelihoods are presented in Tab. 1. The largest log-likelihood among direct competitors is in bold for each dataset. We furthermore tested for statistical significance within the group RAT-SPN, LearnSPN, and ID-SPN[4] where we denote with ∘ results which are not significantly worse than the best one (according to a two-sample t-test, $p = 0.05$).

[3] OBMM is the only other approach, which also employs random structures. However, it does not compile to computation graphs and does not make use of deep neural learning techniques.

[4] For LearnSPN-RGVS, LearnSPN+CCCP, and RandSPN+OBMM we unfortunately had no sample-wise results, so no significance test could be conducted.

| dataset | domain | C | #feat. | #train | #val. | #test |
|---|---|---|---|---|---|---|
| mnist | image | 10 | 784 | 54k | 6k | 10k |
| f-mnist | image | 10 | 784 | 54k | 6k | 10k |
| imdb | text | 2 | 200 | 20k | 5k | 25k |
| theorem | logic | 6 | 51 | 3670 | 1224 | 1224 |
| 20ng | text | 20 | 50 | 13568 | 1508 | 3770 |
| higgs | physics | 2 | 28 | 9M | 1M | 1M |
| wine | chem. | 2 | 11 | 3899 | 1299 | 1299 |

Table 2: Overview of classification datasets.

| dataset | GMM | RAT-SPN | MLPd | MLP+ |
|---|---|---|---|---|
| mnist | 97.37 | ∘**98.29** | 98.05 | ∘98.52 |
| f-mnist | 88.08 | 89.43 | **89.89** | 90.63 |
| imdb | ∘75.65 | ∘**75.90** | ∘75.72 | ∘75.83 |
| theorem | ∘55.64 | ∘55.47 | ∘**57.76** | ∘56.21 |
| 20ng | 47.61 | ∘**48.49** | ∘**48.49** | ∘48.97 |
| higgs | 74.14 | 73.82 | **76.36** | 76.45 |
| wine | ∘77.21 | ∘77.14 | ∘**77.83** | ∘79.45 |

Table 3: Test classification accuracy, best values in bold for each dataset. Results which are not significantly different (according to McNemar's test) from the best are denoted by ∘. Note that MLP+ uses additional training techniques (see text), which make it directly not comparable to the other methods.

The results in Tab. 1 are surprising, as the log-likelihoods of RAT-SPN are often close to the ones of ID-SPN. In fact, ID-SPN is significantly better than RAT-SPN on only 7 out of 20 datasets. moreover, RAT-SPNs are only on 5 datasets more than $5\%$ worse, relative to ID-SPN. Given that RAT-SPNs do not use *any* structure learning at all, while ID-SPN is a highly sophisticated structure learner, the difference is indeed surprisingly small. On three datasets RAT-SPNs even perform better than ID-SPN, although not significantly. Moreover, RAT-SPNs almost consistently outperform RandSPN+OBMM, except on 'msnbc'. On 8 datasets, RandSPN+OBMM performs more than $5\%$ worse, relative to RAT-SPNs. Given that RandSPN+OBMM is the only other approach using random structures, we find that RAT-SPNs establish state-of-the-art for SPNs with random structures. One should note, that this comparison to OBMM is not entirely fair, since RAT-SPNs explore much larger structures, and are also not restricted to trees. However, our hypothesis for this paper was that overparameterized SPNs with simple parameter learning deliver satisfying results. We find that the results in Table 1 confirm this hypothesis.

### 4.2 DISCRIMINATIVE LEARNING: RAT-SPNs ARE COMPETITIVE WITH NEURAL NETS

Next, we evaluated the discriminative performance of RAT-SPNs. This time, the natural competitors are deep neural networks, as discriminative structure learning for SPNs has been unexplored so far[5]. To this end, we apply RAT-SPNs to 7 classification tasks from various domains. Tab. 4 summarizes the characteristics of these datasets, see supplementary for additional details.

Due to their random nature, RAT-SPNs are *domain agnostic*, i.e. they do not have an inductive bias tailored towards any particular type of data, as opposed to e.g. convolutional neural networks for images. Clearly, incorporating convolutional structures in SPNs would be advantageous for 'mnist' and 'fashion-mnist', as demonstrated in [52]. However, the model-agnostic character of RAT-SPNs allows their use in a wider range of problems, and in particular their performance would not degrade if the pixels of '(fashion-)mnist' were scrambled. As input distributions we used Gaussians with variance fixed to 1.

We compared RAT-SPNs to multi-layer perceptrons (MLPs) with rectified linear units, which are one of the most powerful domain-agnostic model class from the deep learning toolbox. We trained MLPs in two variants, namely a standard variant using only dropout (MLPd) – like in RAT-SPNs – and a variant (MLP+) also employing Xavier-initialization [24] and batch normalization [27]. The latter includes two additional training heuristics, which have evolved over decades, while similar techniques for RAT-SPNs are not yet available. Thus, MLPd might serve as a fairer comparison to RAT-SPNs.

For both RAT-SPNs and MLPs, we cross-validated the "depth" (number of hidden layers for MLPs, and split-depth $D$ for RAT-SPNs), and the "width" (number of hidden units for MLPs, and parameters $R$, $S$ and $I$ for RAT-SPNs). Thereby, we first selected suitable ranges for the MLP's hyper-parameters and *then* matched the sizes of the RAT-SPN. Thus, the comparison is fair in terms of considered depth and number of model parameters. The ceomplete hyperparameter configurations are reported in the supplementary.

All models were trained for 200 epochs, optimizing cross-entropy using Adam in its default setting and a batchsize of 100. For regularization, we applied early stopping and dropout-rates $\{0.25, 0.5, 0.75, 1.0\}$, independently for inputs and hidden layers/sum layers. For 'higgs', we only trained one epoch due to the large number of samples, i.e. we effectively considered an on-

[5]With the exceptions of [2], which is actually performing LearnSPN-like generative structure learning on each set of samples belonging to a single class, and Rooshenas and Lowd DACLEARN [51], which focuses on Arithmetic Circuits over discrete domains.

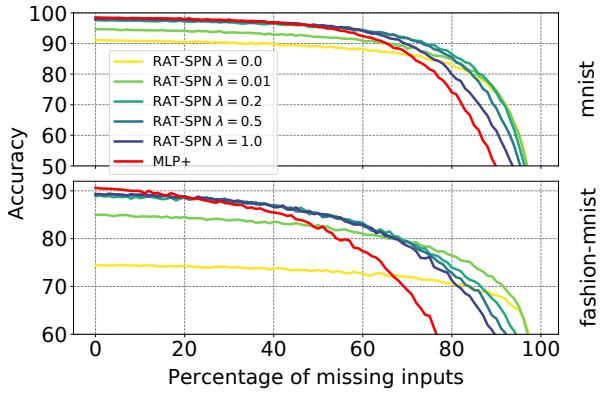


Figure 1: Classification accuracy (y-axis) of hybrid RAT-SPNs and MLP+ as a function of percentage $p$ of missing input features (x-axis), varying from $0\%$ (no features missing) to $99\%$ (almost all features missing) on mnist (top) and fashion-mnist (bottom). For better readability, only the accuracy range 50%-100% (resp. 60%-100%) is shown for mnist (resp. fashion-mnist).

line setting. We further compared to Gaussian mixture models (GMMs) with a massive number of components, namely $1000$, $2000$, $4000$, and $8000$. In this way, GMMs provide a classification baseline for SPNs. The number of components was cross-validated as well as the dropout-rates at the inputs – dropout was applied in similar fashion as for RAT-SPNs. For the covariance matrices, we used the unity matrix.

Tab. 3 summarizes the classification performances on the test sets. We see that RAT-SPNs compare well to MLPd. Out of the 7 dataset, RAT-SPNs win 2 times against MLPd and have one draw (the number of correct examples for 20ng was indeed exactly the same). Moreover, RAT-SPNs are only twice significantly worse than MLP+. We see that GMMs tend to perform slightly better than RAT-SPNs on datasets with few variables. On the datasets with many variables, however, RAT-SPNs perform considerably better. This is consistent with the well-known fact that GMMs do not scale well to high-dimensional spaces. Overall, we see that RAT-SPNs deliver decent classifiers when trained discriminatively. So far, most works on discriminative parameter estimation for SPNs were tailored to images, exploiting either powerful pre-extracted features [22] or using specialized structures [52, 3]. Our results are the first, which investigate the effectiveness of SPNs when trained end-to-end using entirely random structures. We do not only scale SPN training to the regime of deep neural learning, but also demonstrate it to be competitive with deep networks.

Actually, as shown next, there are a number of advantages of employing RAT-SPNs over deep neural networks due to the fact that they represent a *tractable* full joint distribution over inputs $\mathbf{X}$ and class $Y$. Since a purely discriminative model, i.e. optimized only for cross-entropy, is not incentivized to capture the distribution over inputs $\mathbf{X}$ well, we performed hybrid generative-discriminative post-training on our RAT-SPN classifiers. Specifically, we applied Adam for 20 additional epochs, optimizing the hybrid objective (5) for various setting of $0 \leq \lambda \leq 1$. For $\lambda$ close to $0$, we get higher test-likelihoods and lower classification accuracies (generative flavor) than for $\lambda$ close to $1$ (discriminative flavor).

### 4.3 RAT-SPNs ARE ROBUST UNDER MISSING FEATURES

When input features in $\mathbf{X}$ are missing at random, we ideally want to marginalize them [35]. As SPNs allow efficient marginalization, RAT-SPNs are highly suitable to be robust under missing features, especially for smaller $\lambda$ values. To this end, we discard pixels with probability $p$ in the test samples for the mnist and fashion mnist datasets and classify them using RAT-SPNs.

Marginalizing missing features amounts to probabilistic dropout used during training, i.e. simply setting corresponding input distributions to $1$. Similarly, we use dropout at test time to allow MLPs to perform under missing features. Alternatively, missing data can be treated with e.g. k-nearest neighbor imputation. This, however, requires one to store the whole training set and to solve an expensive nearest neighbor search for each test sample.

Fig. 1 summarizes the classification results when varying the fraction of missing features $p$ between $0.0$ and $0.99$. As one can see, RAT-SPNs are more robust than MLP+ using dropout. This effect becomes stronger with smaller $\lambda$, i.e. for models with a "more generative flavor". A particularly interesting choice is $\lambda = 0.2$: here the corresponding RAT-SPN starts with an accuracy of $97.61\%$ for no missing features and degrades very gracefully: for a large fraction of missing features ($> 60\%$) the advantage over MLP+ is dramatic.

### 4.4 RAT-SPNs KNOW WHAT THEY DO NOT KNOW

Besides being robust under missing features, an important feature of (hybrid) generative models is that they are naturally able to detect outliers and peculiarities by monitoring the marginal likelihood over inputs $\mathbf{X}$. Our aim in this section is to demonstrate empirically that RAT-SPNs readily provide well-calibrated uncertainties. We do this first for classification, where hybrid RAT-SPNs offer a principled probabilistic mechanism to deal with uncer-

Figure 2: Examples of outliers (first row) and inliers (third row) for 'mnist' and 'fashion-mnist', for each class. Left column samples were classified correctly, while right ones incorrectly. See supplementary for image in higher resolution and further evaluation.

tainty, something that is clearly not accessible to classic deep neural classifiers. Second, we investigate the ability of generative SPNs to perform anomaly detection on certain image datasets using the likelihood as an outlier score. Very recently, [39, 11] have demonstrated a variety of deep generative models to fail at this task, assigning a higher likelihood to the out-of-domain set than to their training set.

For the classification setting, we evaluated the likelihoods on the test set for both 'mnist' and 'fashion-mnist', using the respective RAT-SPN post-trained with $\lambda = 0.2$. For illustrative purposes, we divided the test samples into correctly and incorrectly classified ones. From both groups, we selected two examples for each class, namely the one with the lowest input probability (outlier) and the one with the highest input probability (inlier). This yielded 4 groups of 10 samples each: outlier/correct, outlier/incorrect, inlier/correct, inlier/incorrect. These samples are shown in Fig. 6 (a higher resolution version is provided in the supplementary).

Albeit qualitative, these results are interesting and important: For 'mnist', one can visually confirm that the outlier digits are indeed peculiar, both the correctly and the incorrectly classified ones. For instance, in the outlier/incorrect group the '0' and '3' have been apparently cut off during pre-processing, the '6' is not recognizable. In the inlier/incorrect group we have rather ambiguous examples, which seems to be the major cause of misclassification. This is reflected by the fact that the predictive uncertainty (cross-entropy of the class distribution) was highest in this group, and that in 8 out of 10 cases the correct class had the second highest probability (see supplementary). Similar results hold for 'fashion-mnist'.

For a more quantitative analysis, we used a variant of *transfer testing* proposed by Bradshaw *et al.* [7]. This technique is quite simple: we feed a classifier trained on one domain (e.g. 'mnist') with examples from a related but different domain, e.g. street view house numbers ('svhn') [40] or the handwritten digits of 'semeion' [9], converted to 'mnist' format ($28 \times 28$ pixels, grey scale).

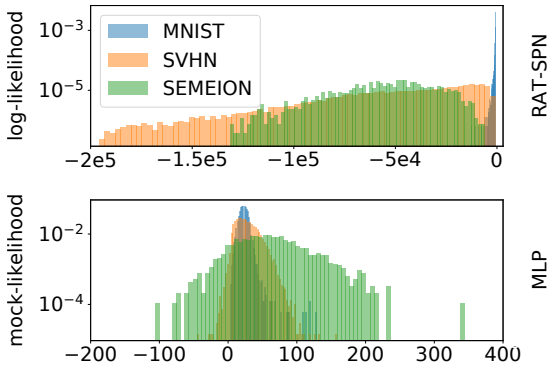


Figure 3: Histograms of test log-likelihoods for 'mnist', 'svhn' and 'semeion' data for RAT-SPN (top) and corresponding computations performed for MLP+ ("mock-likelihood") (bottom). Both models were trained on 'mnist'. The likelihoods of RAT-SPNs yield a strong signal whether a sample is in-domain or out-of-domain.

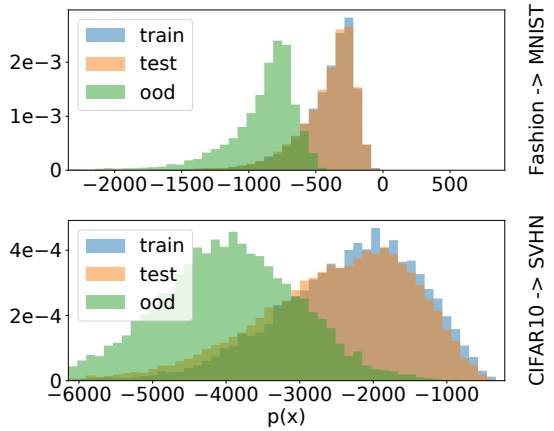


Figure 4: Histograms the log-likelihoods obtained from a RAT-SPN on the set it was trained on ( 'fashion-mnist' on top and 'cifar10' on bottom, in blue), the corresponding test set (orange) and an out-of-domain dataset ('mnist' on top and 'svhn' on bottom, in green). Differently from VAEs and GLOW, see Fig.2 in [39], RAT-SPNs *do not* assign higher likelihood to out-of-domain samples.

While we would expect that most classifiers perform poorly in such setting, an AI system *should* be aware that it is confronted with out-of-domain data. While Bradshaw *et al.* applied this technique to output uncertainties, it is clearly also applicable to *input uncertainties*, i.e. the marginal probability of features $\mathbf{X}$.

Fig. 3(top) shows histograms of the log-probabilities over inputs for the RAT-SPN post-trained with $\lambda = 0.2$, when fed with 'mnist' test data (in-domain), 'svhn' test data (out-of-domain) and 'semeion' (out-of-domain). The likelihood histograms provide a strong

signal whether a sample comes from in-domain or out-of-domain. In fact, the samples from 'mnist' and 'semeion' can be perfectly discriminated, i.e. the highest input probability in 'semeion' is smaller than the lowest input probability in 'mnist'. The samples of 'mnist' and 'svhn' overlap by less than $1\%$. Consequently, RAT-SPNs — and other tractable joint probability models — have an extra communication channel to inform us whether we ought to trust their predictions.

However, does this result indeed stem from the fact that we model a full joint distribution, or merely from the fact that we average outputs of a classifier? Thus, as a sanity check, we performed the likewise computations our trained MLP+. One might suspect that the result, although not interpretable as log-probability, still yields a decent signal to detect outliers. In need of a name for this exotic quantity, we name it *mock-likelihood*. Fig. 3(bottom) shows histograms of this mock-likelihood: although more spread, histograms for out-of-domain data are highly overlapping and do not yield a clear signal for out-of-domain vs. in-domain.

We apply the same line of reasoning for outlier detection in the generative case, investigating if RAT-SPNs are susceptible to the same "likelihood mirage" that affects several deep generative models such as VAEs, ARDEs and NFs: It has been recently noted [11, 39] that samples from certain test image datasets are not only hard to be recognized as out-of-domain, but are consistently deemed to be even more likely than in-domain samples. In [39] this has been reported for VAEs, ARDEs like PixelCNNs [55], and NFs such as GLOW [31] when they are trained on image data that is clearly – at least for a human – very different from the target test. This behavior is definitely unexpected, since VAEs, PixelCNNs and GLOW, differently from the MLPs we employed for the classification experiments, are trained to directly maximise the likelihood, which has classically been considered a proper score for anomaly detection [10, 25].

We replicate the experimental setting of [39] by training a generative RAT-SPN on the training sets of 'fashion-mnist' and 'cifar10'. We then evaluate the likelihood of in-domain test samples (belonging to the same dataset) and of out-of-domain samples coming from 'mnist' and 'svhn', respectively. Fig. 4 reports the histogram of the log-likelihoods RAT-SPNs used to score train and test in-domain and out-of-domain samples for 'fashion-mnist' $\rightarrow$ 'mnist' (top) and 'cifar10' $\rightarrow$ 'svhn' (bottom).

Differently from VAE, PixelCNN and GLOW (please refer to [39] for corresponding plots), RAT-SPNs are not assigning higher likelihoods to out-of-domain samples and clearly discriminate among inliers and outliers. This is evident for 'mnist' against 'fashion-mnist' and slightly less prominent in the other case where 'svhn' likelihood histogram overlaps slightly more with 'cifar10' ones. In any case, this clearly highlights the superior ability of RAT-SPNs to properly calibrate uncertainties when compared to current deep generative models based on neural networks, which fall prey to the "likelihood mirage".

# 5 CONCLUSION

We have established a link between tractable probabilistic models and deep neural learning, and used the link to demonstrate that tractable models get surprisingly far without detailed structure learning. Specifically, the simple and scalable approach, called RAT-SPN, to construct a random but valid SPN structure, tensorize it, and combine it with simple training mechanisms like soft EM or Adam delivers results comparable to state-of-the-art, both in the generative and the discriminative setting. This represents a *tremendous simplification* of learning SPNs and in turn paves the way to a wider use of tractable probabilistic models in the deep learning community.

By implementing RAT-SPNs in Tensorflow, we automatically make use of GPU computations leading to considerable speed-ups compared to traditional SPN learning on CPUs. For example, one epoch on 'mnist' takes roughly a minute for a RAT-SPN with depth 2 and 1.2M parameters, using a GTX 1080Ti. This is a speedup of 45 X compared to a single CPU. However, a comparable MLP with 1.2M parameters only needs slightly more than 1 sec/epoch, but this is not surprising. MLPs rely on highly parallelized matrix multiplications and cheap non-linearities. On the other hand, RAT-SPNs bring enhanced sparsity in the weight matrix to establish consistency across any marginals and, therefore, make the computation less efficient on GPUs. Moreover, they employ expensive log-sum-exp computations, used to avoid numerical underflow. To speed-up RAT-SPNs, one can approximate them in each region with a sparsified variant. This avoids to generate all cross-products reducing the number of operands involved. One could also perform operations in the linear domain, together with a smart rescaling approach to avoid numerical underflow. Furthermore, we are currently investigating approaches using specialized hardware such as FPGAs for SPNs.

Overall, the ideas and results presented in this paper are intriguing directions for probabilistic deep learning. As demonstrated, SPNs are capable connectionist models with additional advantages like calibrated anomaly detection, treatment of missing features, or more generally, the power of tractable probabilistic inference. Exploring these feature jointly with deep neural networks e.g. as callbrated loss layers is the most promising avenue for future work.

## References


[1] Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

[2] T. Adel, D. Balduzzi, and A. Ghodsi. Learning the structure of sum-product networks via an SVD-based algorithm. In *UAI*, 2015.

[3] M. R. Amer and S. Todorovic. Sum product networks for activity recognition. *IEEE transactions on pattern analysis and machine intelligence*, 38(4):800–813, 2016.

[4] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5:4308, 2014.

[5] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3:993–1022, 2003.

[6] G. Bouchard and B. Triggs. The Trade-Off between Generative and Discriminative Classifiers. In *COMPSTAT*, pages 721–728, 2004.

[7] J. Bradshaw, A. Matthews, and Z. Ghahramani. Adversarial examples, uncertainty, and transfer testing robustness in gaussian process hybrid deep networks. *preprint arXiv*, 2017. arxiv.org/abs/1707.02476.

[8] J. P. Bridge, S. B. Holden, and L. C. Paulson. Machine learning for first-order theorem proving. *Journal of Automated Reasoning*, 53(2):141–172, 2014.

[9] M. Buscema. *MetaNet*: The Theory of Independent Judges*, volume 33. Taylor & Francis, 02 1998.

[10] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[11] Hyunsun Choi and Eric Jang. Generative ensembles for robust anomaly detection. *arXiv preprint arXiv:1810.01392*, 2018.

[12] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 2009.

[13] A. Darwiche. A differential approach to inference in Bayesian networks. *Journal of the ACM*, 50(3):280–305, 2003.

[14] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[15] A. Dennis and D. Ventura. Learning the architecture of sum-product networks using clustering on variables. In *Proceedings of NIPS*, pages 2042–2050, 2012.

[16] A. Dennis and D. Ventura. Greedy structure search for sum-product networks. In *IJCAI*, pages 932–938, 2015.

[17] A. Dennis and D. Ventura. Online structure-search for sum-product networks. In *Proceedings of ICMLA*, pages 155–160, 2017.

[18] N. Di Mauro, F. Esposito, F. G. Ventola, and A. Vergari. Sum-product network structure learning by efficient product nodes discovery. *Intelligenza Artificiale*, 12(2):143–159, 2018.

[19] N. Di Mauro, A. Vergari, T. Basile, and F. Esposito. Fast and accurate density estimation with extremely randomized cutset networks. In *Proceedings of ECML/PKDD*, pages 203–219, 2017.

[20] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[21] C. Févotte and J. Idier. Algorithms for nonnegative matrix factorization with the $\beta$-divergence. *Neural computation*, 23(9):2421–2456, 2011.

[22] R. Gens and P. Domingos. Discriminative learning of sum-product networks. In *Proceedings of NIPS*, pages 3248–3256, 2012.

[23] R. Gens and P. Domingos. Learning the structure of sum-product networks. *Proceedings of ICML*, pages 873–880, 2013.

[24] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of AISTATS*, pages 249–256, 2010.

[25] Markus Goldstein and Seiichi Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PloS one*, 11(4):e0152173, 2016.

[26] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Proceedings of NIPS*, pages 2672–2680, 2014.

[27] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Prooceedings of ICML*, pages 448–456, 2015.

[28] P. Jaini, A. Ghose, and P. Poupart. Prometheus: Directly learning acyclic directed graph structures for sum-product networks. In *PGM*, 2018.

[29] T. Joachims. A probabilistic analysis of the rocchio algorithm with TFIDF for text categorization. Technical report, 1996.

[30] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. In *ICLR*, 2014. arXiv:1312.6114.

[31] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10236–10245, 2018.

[32] H. Larochelle and I. Murray. The neural autoregressive distribution estimator. In *Proceedings of AISTATS*, pages 29–37, 2011.

[33] Y. LeCun, C. Cortes, and C. J. C. Burges. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

[34] Y. Liang, J. Bekker, and G. Van den Broeck. Learning the structure of probabilistic sentential decision diagrams. In *Proceedings of UAI*, 2017.

[35] R. J. A. Little and D. B. Rubin. *Statistical analysis with missing data*, volume 333. John Wiley & Sons, 2014.

[36] D. Lowd and A. Rooshenas. Learning Markov Networks with Arithmetic Circuits. *Proceedings of AISTATS*, pages 406–414, 2013.

[37] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proceedings of ACL*, pages 142–150, 2011.

[38] A. Molina, A. Vergari, N. Di Mauro, S. Natarajan, F. Esposito, and K. Kersting. Mixed sum-product networks: A deep architecture for hybrid domains. In *Proceedings of AAAI*, 2018.

[39] Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. Do deep generative models know what they don't know? *International Conference on Learning Representations (ICLR)*, 2019.

[40] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[41] R. Peharz, B. Geiger, and F. Pernkopf. Greedy part-wise learning of sum-product networks. In *Proceedings of ECML/PKDD*, volume 8189, pages 612–627, 2013.

[42] R. Peharz, R. Gens, and P. Domingos. Learning selective sum-product networks. In *ICML-LTPM Workshop*, 2014. online: https://sites.google.com/site/ltpm2014/.

[43] R. Peharz, R. Gens, F. Pernkopf, and P. Domingos. On the latent variable interpretation in sum-product networks. *IEEE TPAMI*, 39(10):2030–2044, 2017.

[44] R. Peharz, S. Tschiatschek, F. Pernkopf, and P. Domingos. On theoretical properties of sum-product networks. In *Proceedings of AISTATS*, pages 744–752, 2015.

[45] H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *Proceedings of UAI*, pages 337–346, 2011.

[46] T. Rahman and V. Gogate. Merging strategies for sum-product networks: From trees to graphs. In *UAI*, 2016.

[47] A. Rashwan, H. Zhao, and P. Poupart. Online and distributed bayesian moment matching for parameter learning in sum-product networks. In *AISTATS*, pages 1469–1477, 2016.

[48] Abdullah Rashwan, Pascal Poupart, and Chen Zhitang. Discriminative training of sum-product networks by extended baum-welch. In *International Conference on Probabilistic Graphical Models*, pages 356–367, 2018.

[49] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of ICML*, pages 1278–1286, 2014.

[50] A. Rooshenas and D. Lowd. Learning Sum-Product Networks with Direct and Indirect Variable Interactions. *ICML – JMLR W&CP*, 32:710–718, 2014.

[51] Amirmohammad Rooshenas and Daniel Lowd. Discriminative structure learning of arithmetic circuits. In *AISTATS*, 2016.

[52] O. Sharir, R. Tamari, N. Cohen, and A. Shashua. Tractable generative convolutional arithmetic circuits. *arXiv preprint arXiv:1610.04167*, 2016.

[53] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[54] M. Trapp, R. Peharz, M. Skowron, T. Madl, F. Pernkopf, and R. Trappl. Structure inference in sum-product networks using infinite sum-product trees. In *NIPS Workshop on Practical Bayesian Nonparametrics*, 2016.

[55] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. In *Proceedings of ICML*, pages 1747–1756, 2016.

[56] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, pages 4790–4798, 2016.

[57] A. Vergari, N. Di Mauro, and F. Esposito. Simplifying, regularizing and strengthening sum-product network structure learning. In *Proceedings of ECML/PKDD*, pages 343–358. Springer, 2015.

[58] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[59] H. Zhao, T. Adel, G. Gordon, and B. Amos. Collapsed variational inference for sum-product networks. In *Proceedings of ICML*, pages 1310–1318, 2016.

## Appendix A Building RAT-SPNs

**Algorithm 2** Construct SPN from Region Graph

**procedure** CONSTRUCTSPN($\mathcal{R}, C, S, I$)
  Make empty SPN
  **for** $\mathbf{R} \in \mathcal{R}$ **do**
    **if** $\mathbf{R}$ is a leaf region **then**
      Equip $\mathbf{R}$ with $I$ distribution nodes
    **else if** $\mathbf{R}$ is the root region **then**
      Equip $\mathbf{R}$ with $C$ sum nodes
    **else**
      Equip $\mathbf{R}$ with $S$ sum nodes
  **for** $\mathcal{P} = \{\mathbf{R}_1, \mathbf{R}_2\} \in \mathcal{R}$ **do**
    Let $\mathbf{N}_{\mathbf{R}}$ be the nodes for region $\mathbf{R}$
    **for** $\mathsf{N}_1 \in \mathbf{N}_{\mathbf{R}_1}, \mathsf{N}_2 \in \mathbf{N}_{\mathbf{R}_2}$ **do**
      Introduce product $\mathsf{P} = \mathsf{N}_1 \times \mathsf{N}_2$
      Let $\mathsf{P}$ be a child for each $\mathsf{N} \in \mathbf{N}_{\mathbf{R}_1 \cup \mathbf{R}_2}$
  **return** SPN

**Algorithm 3** Random Region Graph

**procedure** RANDOMREGIONGRAPH($\mathbf{X}, D, R$)
  Create an empty region graph $\mathcal{R}$
  Insert $\mathbf{X}$ in $\mathcal{R}$
  **for** $r = 1 \dots R$ **do**
    SPLIT($\mathcal{R}, \mathbf{X}, D$)

**procedure** SPLIT($\mathcal{R}, \mathbf{R}, D$)
  Draw balanced partition $\mathcal{P} = \{\mathbf{R}_1, \mathbf{R}_2\}$ of $\mathbf{R}$
  Insert $\mathbf{R}_1, \mathbf{R}_2$ in $\mathcal{R}$
  Insert $\mathcal{P}$ in $\mathcal{R}$
  **if** $D > 1$ **then**
    **if** $|\mathbf{R}_1| > 1$ **then** SPLIT($\mathcal{R}, \mathbf{R}_1, D-1$)
    **if** $|\mathbf{R}_2| > 1$ **then** SPLIT($\mathcal{R}, \mathbf{R}_2, D-1$)

Recall from the main paper, that given a set of RVs $\mathbf{X}$, a *region* $\mathbf{R}$ is defined as any non-empty subset of $\mathbf{X}$. Given any region $\mathbf{R}$, a *$K$-partition* $\mathcal{P}$ of $\mathbf{R}$ is a collection of $K$ non-empty, non-overlapping subsets $\mathbf{R}_1, \dots, \mathbf{R}_K$ of $\mathbf{R}$, whose union is again $\mathbf{R}$, i.e. $\mathcal{P} = \{\mathbf{R}_1, \dots, \mathbf{R}_K\}$, $\forall k\colon \mathbf{R}_k \neq \emptyset$, $\forall k \neq l\colon \mathbf{R}_k \cap \mathbf{R}_l = \emptyset$, $\bigcup_k \mathbf{R}_k = \mathbf{R}$. We consider only 2-partitions, which causes all product nodes in our SPNs to have exactly two children. This assumption, frequently made in the SPN literature, simplifies SPN design and seems not to impair performance.

A *region graph* $\mathcal{R}$ over $\mathbf{X}$ is a connected DAG whose nodes are regions and partitions such that i) there is exactly one region $\mathbf{R} = \mathbf{X}$ without parents (i.e. $\mathbf{X}$ is the *root region*), ii) all leaves of $\mathcal{R}$ are regions, iii) all children of regions are partitions and all children of partitions are regions (i.e. $\mathcal{R}$ is bipartite), iv) if $\mathcal{P}$ is a child of $\mathbf{R}$, then $\bigcup_{\mathbf{R}' \in \mathcal{P}} \mathbf{R}' = \mathbf{R}$ and v) if $\mathbf{R}$ is a child of $\mathcal{P}$, then $\mathbf{R} \in \mathcal{P}$. From this definition it follows that a region graph dictates a hierarchical partitioning of the overall scope $\mathbf{X}$.

Now, given a region graph $\mathcal{R}$, we can easily construct a corresponding SPN as follows: Populate each leaf-region with a collection of $I$ input distributions, and all other regions with a collection of sum nodes. For the root region we spend $C$ sum nodes, and for all internal regions we spend $S$ sum nodes. Finally, take all cross-products of nodes which are co-children of a partition, and connect these product as children of all sums in the parent region of this partition. Pseudo-code for this procedure is given in Algorithm 2. An example for an RAT-SPN by applying Algorithm 3 (Random Region Graph), followed by Algorithm 2, is given in Figure 5, for hyper-parameters $R = 2$, $D = 2$, $I = 2$, $S = 2$, $C = 3$.

## Appendix B Classification Datasets

We performed classification experiments on

- 'mnist', a well-known digit classification dataset [33]
- 'fashion-mnist', an in-place substitute for 'mnist', with the goal to classify fashion items rather than digits [58]
- 'imdb', a dataset for binary sentiment analysis of popular movie reviews [37]
- '20ng', a dataset of newsgroup posts belonging to 20 different topics [29]
- 'theorem', a task where a suitable heuristic shall be selected for automatic theorem proofing [8]
- 'higgs', a task to detect whether Higgs boson was generated during a high energy particle collision [4]
- 'wine', a dataset where wine quality shall be predicted from various chemical features [12]

An overview of various characteristics of these datasets is shown in Table 4.

For '(fashion-)mnist', we removed pixels with very low variance (in particular, where the pixel's variance is smaller than $0.001$ times the average variance). For 'mnist', we were left with $629$ pixels and for 'fashion-mnist' with $775$ pixels, out of the originally $784$ pixels.

For '20ng', the text was pre-processed into a bag-of-words representation by keeping the top $1000$ most relevant words according to their Tf-IDF. Then, $50$ topics were extracted by LDA [5] and employed as the new feature representation for classification.

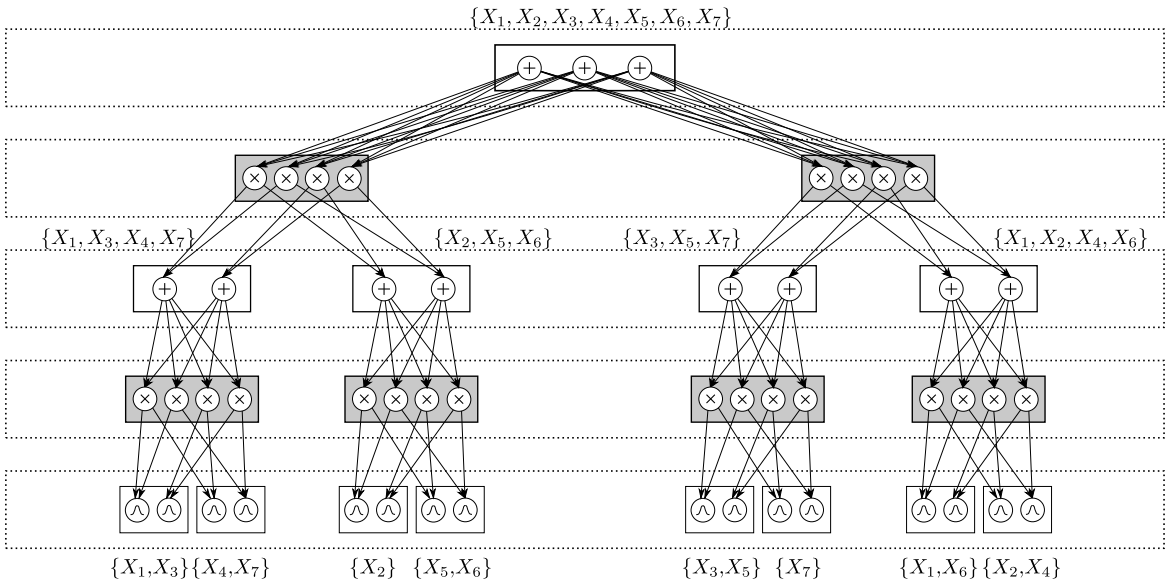


Figure 5: An example RAT-SPN for $R = 2$, $D = 2$, $I = 2$, $S = 2$, $C = 3$.

| dataset | domain | #feat. | #train | # val. | # test |
|---|---|---|---|---|---|
| mnist | image | 784 | 54k | 6k | 10k |
| f-mnist | image | 784 | 54k | 6k | 10k |
| imdb | text | 200 | 20k | 5k | 25k |
| theorem | logic | 51 | 3670 | 1224 | 1224 |
| 20ng | text | 50 | 13568 | 1508 | 3770 |
| higgs | physics | 28 | 9M | 1M | 1M |
| wine | chemistry | 11 | 3899 | 1299 | 1299 |

Table 4: Overview of classification datasets.

For 'imdb', English stopwords have been removed, and only the 1000 most frequent words have been kept to compute the TF-IDF representation of the training document collection. Over this representation, we extracted 200 topics with Non-Negative Matrix Factorization optimizing the KL-divergence [21] and a L2-regularization term with coefficient $\alpha = 0.1$ via multiplicative updates for 1000 iterations after a random initialization.

For 'theorem', 'higgs' and 'wine' we used the provided raw features. For 'wine' we consolidated the two subsets for red and white wine.

Subsequently, we performed for all datasets zero-mean unit-variance normalization. For all classifiers, we used exactly the same pre-processing.

# Appendix C Selection of Hyper-Parameters

RAT-SPNs have 5 structural hyper-parameters, i.e.

- *split-depth* $D$
- *number of repetitions* $R$
- *number of input distributions* $I$
- *number of root nodes* $C$
- *number of sum nodes per inner region* $S$

See the main paper and the previous section (Algorithm 3 and Algorithm 2) for details concerning these hyper-parameters.

In order to adequately set these hyper-parameters, we might target at a particular number of parameters, i.e. sum-weights and parameters of input distributions. Recall from the main paper that the number of sum-weights is given as

$$W_{\mathsf{S}} = \begin{cases} RCI^2 & \text{if } D = 1 \\ R\left(CS^2 + (2^{D-1} - 2)S^3 + 2^{D-1}SI^2\right) & \text{if } D > 1. \end{cases} \tag{6}$$

Similarly, we can count the number of parameters of input distributions, which we assume to factorize into univariate distributions. Since the exponential growth of number of input regions is exactly compensated by the

Table 5: Choice of hyper-parameters $R$, $I$, and $S$ for generative learning. For $D = 1$, hyper-parameter $S$ has no effect.

| $D$ | $R$ | $I$ | $S$ | $W_{\mathsf{S}}$ |
|---|---|---|---|---|
| | 10 | 10 | - | 1000 |
| 1 | 25 | 20 | - | 10000 |
| | 50 | 45 | - | 101250 |
| | 4 | 5 | 5 | 1100 |
| 2 | 10 | 8 | 8 | 10880 |
| | 15 | 15 | 15 | 104625 |
| | 3 | 5 | 3 | 1089 |
| 3 | 10 | 6 | 5 | 9950 |
| | 16 | 10 | 10 | 97600 |
| | 3 | 3 | 3 | 1161 |
| 4 | 6 | 5 | 5 | 10650 |
| | 10 | 10 | 8 | 95360 |

exponential decrease of scope-size per input region, the total number of parameters for the input distributions is given as

$$W_{\mathsf{D}} = RI|\mathbf{X}|P, \tag{7}$$

where $P$ is the number of parameters per univariate distribution.

### C.1 Hyper-Parameters for Generative Learning

For generative learning, we cross-validated the split-depth $D \in \{1, 2, 3, 4\}$ and selected, for each $D$, settings for $R$, $I$, $S$ in order to yield RAT-SPNs approximately with $10^3$, $10^4$ and $10^5$ sum-weights. We set $C = 1$, since we are estimating a single density over $\mathbf{X}$. Our particular choice for $R$, $I$, $S$ is depicted in Table 5. This choice was found by trying some combinations of $R$, $I$, $S$ in (7), until $W_{\mathsf{S}}$ was close to the desired value. We kept the values rather balanced, slightly preferring larger values for $R$ and $I$, since $W_{\mathsf{S}}$ grows quickest with $S$. However, the particular choice for $R$, $I$, $S$ was not tuned to any data – only $W_{\mathsf{S}}$ was cross-validated.

### C.2 Hyper-Parameters for Discriminative Learning

We cross-validated the number of hidden layers $L$ and number of hidden units $H$ in our trained MLPs as depicted in Table 6. We used varying ranges for $L$ and $H$, since the employed datasets have rather distinct sizes. In order to ensure a fair comparison, we selected hyper-parameters $D$, $R$, $I$ and $S$ in order that the number of parameters in RAT-SPNs match the number of parameters in MLPs, see Table 7. Similar as for the generative case, the particular choice of $R$, $I$ and $S$ (for each $D$) was made before running any experiments. Thus, only the number of parameters and split-depth $D$ was cross-validated.

## Appendix D Detecting Outliers

Besides being robust against under features, an important feature of (hybrid) generative models is that they are naturally able to detect outliers and peculiarities by monitoring the marginal likelihood over inputs $\mathbf{X}$. To this end, we evaluated the likelihoods on the test set for both 'mnist' and 'fashion-mnist', using the respective RAT-SPN post-trained with $\lambda = 0.2$. For illustrative purposes, we divided the test samples into correctly and incorrectly classified ones. From both groups, we selected two examples for each class, namely the one with the lowest input probability (outlier) and the one with the highest input probability (inlier). This yields 4 groups of 10 samples each: outlier/correct, outlier/incorrect, inlier/correct, inlier/incorrect. These samples are shown in Figure 6.

Furthermore, Tables 8 and 9 show detailed class posteriors for the incorrect samples in 'mnist' and 'fashion-mnist', respectively. For both datasets we see that in the inlier/incorrect group, ambiguity seems to be the major cause for miss-classification. For 'mnist', we see that for the inlier/incorrect group, the correct class gets 8 out of 10 times the second highest probability, while for the outlier/incorrect group this happens only 4 out of 10 times. For 'fashion-mnist', we see that for the inlier/incorrect group, the correct class gets 6 out of 10 times the second highest probability, while for the outlier/incorrect group this happens only 3 out of 10 times. This is also reflected in the predictive uncertainties, measured as cross-entropy of the class-posterior. In particular, we have for 'mnist':

- outlier/correct: CE = 0.031
- inlier/correct: CE = 0.031
- outlier/incorrect: CE = 0.125
- inlier/incorrect: CE = 0.301

For 'fashion-mnist', we have:

- outlier/correct: CE = 0.023
- inlier/correct: CE = 0.019
- outlier/incorrect: CE = 0.173
- inlier/incorrect: CE = 0.432

Table 6: Choice of hyper-parameters for MLPs, number of hidden layers $L$ and number of hidden units $H$, for the employed classification datasets.

| dataset | $L$ | $H$ | #params |
|---|---|---|---|
| (f-)mnist | 1 | $\{100, 250, 500, 1000, 2000\}$ | 64k, 160k, 320k, 640k, 1.28M |
| | 2 | $\{100, 250, 500, 1000, 2000\}$ | 74k, 221k, 574k, 1.60M, 5.25M |
| | 3 | $\{100, 250, 500, 1000, 2000\}$ | 84k, 286k, 821k, 2.64M, 9.28M |
| | 4 | $\{100, 250, 500, 1000, 2000\}$ | 94k, 348k, 1.07M, 3.64M, 13.29M |
| imdb | 1 | $\{100, 250, 500, 1000, 2000\}$ | 20k, 51k, 102k, 203k, 406k |
| | 2 | $\{100, 250, 500, 1000, 2000\}$ | 30k, 114k, 352k, 1.2M, 4.4M |
| | 3 | $\{100, 250, 500, 1000, 2000\}$ | 41k, 176k, 603k, 2.2M, 8.41M |
| theorem | 1 | $\{100, 250, 500, 1000\}$ | 6k, 15k, 29k, 58k |
| | 2 | $\{100, 250, 500, 1000\}$ | 16k, 77k, 280k, 1.05M |
| | 3 | $\{100, 250, 500, 1000\}$ | 26k, 140k, 530k, 2.06M |
| 20ng | 1 | $\{100, 250, 500, 1000\}$ | 7k, 18k, 36k, 71k |
| | 2 | $\{100, 250, 500, 1000\}$ | 17k, 81k, 286k, 1.07M |
| | 3 | $\{100, 250, 500, 1000\}$ | 27k, 143k, 536k, 2.07M |
| higgs | 1 | $\{100, 250, 500, 1000\}$ | 3k, 8k, 16k, 31k |
| | 2 | $\{100, 250, 500, 1000\}$ | 13k, 71k, 266k, 1.03M |
| | 3 | $\{100, 250, 500, 1000\}$ | 23k, 133k, 517k, 2.03M |
| wine | 1 | $\{100, 250, 500\}$ | 1k, 4k, 7k |
| | 2 | $\{100, 250, 500\}$ | 12k, 66k, 258k |
| | 3 | $\{100, 250, 500\}$ | 22k, 129k, 508k |

Table 7: Choice of hyper-parameters for RAT-SPNs, split-depth $D$, number of repetitions $R$, number of input distributions $I$ and number of sum nodes per inner region $S$, matched to the number of parameters in Table 6. For $D = 1$, hyper-parameter $S$ has no effect.

| datset | $D$ | $(R, I, S)$ | #params |
|---|---|---|---|
| (f-)mnist | 1 | $\{(9, 10, -), (14, 15, -), (19, 20, -), (29, 25, -), (40, 33, -)\}$ | 66k, 164k, 315k, 637k, 1.27M |
| | 2 | $\{(8, 10, 10), (12, 15, 15), (19, 20, 18), (30, 25, 25), (40, 37, 35)\}$ | 74k, 221k, 574k, 1.6M, 5.28M |
| | 3 | $\{(10, 8, 8), (12, 14, 12), (15, 20, 18), (30, 25, 20), (40, 35, 30)\}$ | 87k, 277k, 844k, 2.57M, 9.28M |
| | 4 | $\{(5, 10, 9), (10, 15, 10), (14, 20, 14), (28, 20, 20), (40, 30, 26)\}$ | 93k, 344k, 1.06M, 3.6M, 12.73M |
| imdb | 1 | $\{(9, 10, -), (15, 15, -), (21, 20, -), (30, 26, -), (40, 36, -)\}$ | 20k, 52k, 101k, 197k, 392k |
| | 2 | $\{(10, 8, 8), (14, 14, 12), (20, 20, 16), (30, 26, 25), (40, 38, 35)\}$ | 28k, 109k, 346k, 1.21M, 4.45M |
| | 3 | $\{(10, 8, 7), (15, 14, 9), (20, 18, 15), (30, 23, 22), (40, 35, 30)\}$ | 42k, 172k, 605k, 2.2M, 8.39M |
| theorem | 1 | $\{(10, 5, -), (13, 7, -), (17, 9, -), (20, 12, -)\}$ | 4k, 8k, 16k, 30k |
| | 2 | $\{(15, 6, 5), (18, 10, 10), (24, 15, 15), (40, 20, 20)\}$ | 12k, 56k, 213k, 777k |
| | 3 | $\{(13, 7, 5), (17, 10, 10), (21, 15, 15), (40, 20, 19)\}$ | 23k, 121k, 470k, 1.89M |
| 20ng | 1 | $\{(10, 5, -), (13, 7, -), (17, 9, -), (20, 12, -)\}$ | 8k, 17k, 35k, 70k |
| | 2 | $\{(15, 6, 5), (18, 10, 10), (24, 15, 15), (40, 20, 20)\}$ | 17k, 81k, 288k, 1M |
| | 3 | $\{(13, 7, 5), (17, 10, 10), (21, 15, 15), (40, 20, 19)\}$ | 27k, 145k, 536k, 2.09M |
| higgs | 1 | $\{(10, 7, -), (16, 10, -), (20, 14, -), (23, 20, -)\}$ | 3k, 8k, 16k, 31k |
| | 2 | $\{(10, 10, 5), (15, 14, 10), (20, 20, 15), (40, 25, 20)\}$ | 13k, 68k, 260k, 1.06M |
| | 3 | $\{(9, 10, 5), (16, 12, 10), (24, 15, 15), (40, 20, 20)\}$ | 23k, 133k, 507k, 1.97M |
| wine | 1 | $\{(5, 10, -), (8, 12, -), (13, 14, -)\}$ | 2k, 3k, 7k |
| | 2 | $\{(5, 10, 10), (10, 15, 14), (20, 20, 15)\}$ | 12k, 69k, 253k |
| | 3 | $\{(9, 10, 5), (11, 15, 10), (20, 20, 13)\}$ | 22k, 125k, 515k |

Figure 6: Examples of outliers (lowest input probability in test set) and inliers (highest input probability) for 'mnist' and 'fashion-mnist', for each class. The respective top row shows the outliers, the bottom row the inliers. All samples on the left hand side were classified correctly, all samples on the right hand side were classified incorrectly.

Table 8: Class posteriors for incorrect outliers and incorrect inliers on 'mnist'.

| | outlier/incorrect | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| label | $p(C \mid \mathbf{X})$ | | | | | | | | | | CE | LL |
| 0 | 0.288 | 0.000 | 0.000 | 0.000 | 0.712 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.601885 | -1252.031372 |
| 1 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000230 | -912.313721 |
| 2 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000177 | -9215.482422 |
| 3 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000606 | -1609.093994 |
| 4 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000182 | -12721.799805 |
| 5 | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000158 | -10736.949219 |
| 6 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000229 | -3916.864502 |
| 7 | 0.000 | 0.000 | 0.000 | 0.971 | 0.000 | 0.000 | 0.000 | 0.029 | 0.000 | 0.000 | 0.132454 | -2431.549316 |
| 8 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.990 | 0.000 | 0.000 | 0.010 | 0.000 | 0.054752 | -2403.748535 |
| 9 | 0.000 | 0.000 | 0.825 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.175 | 0.463652 | -2462.062256 |
| | inlier/incorrect | | | | | | | | | | | |
| label | $p(C \mid \mathbf{X})$ | | | | | | | | | | CE | LL |
| 0 | 0.252 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.748 | 0.000 | 0.000 | 0.000 | 0.564740 | -682.124390 |
| 1 | 0.000 | 0.132 | 0.000 | 0.000 | 0.013 | 0.000 | 0.000 | 0.852 | 0.000 | 0.002 | 0.473643 | -670.625916 |
| 2 | 0.000 | 0.266 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.734 | 0.000 | 0.582508 | -675.940247 |
| 3 | 0.000 | 0.000 | 0.000 | 0.486 | 0.000 | 0.000 | 0.000 | 0.000 | 0.514 | 0.000 | 0.693245 | -677.803772 |
| 4 | 0.000 | 0.000 | 0.000 | 0.000 | 0.006 | 0.000 | 0.000 | 0.000 | 0.000 | 0.994 | 0.035805 | -663.004822 |
| 5 | 0.000 | 0.001 | 0.002 | 0.001 | 0.019 | 0.034 | 0.000 | 0.004 | 0.000 | 0.937 | 0.307368 | -715.435852 |
| 6 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000124 | -669.673096 |
| 7 | 0.000 | 0.970 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.030 | 0.000 | 0.000 | 0.135932 | -660.813416 |
| 8 | 0.000 | 0.000 | 0.998 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.002 | 0.000 | 0.014296 | -685.410889 |
| 9 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.950 | 0.000 | 0.050 | 0.198429 | -658.176208 |

Table 9: Class posteriors for incorrect outliers and incorrect inliers on 'fashion-mnist'.

| | outlier/incorrect | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| label | $p(C \mid \mathbf{X})$ | | | | | | | | | | CE | LL |
| 0 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.003652 | -4592.819336 |
| 1 | 0.001 | 0.000 | 0.003 | 0.000 | 0.000 | 0.000 | 0.996 | 0.000 | 0.000 | 0.000 | 0.030717 | -1112.185669 |
| 2 | 0.001 | 0.000 | 0.000 | 0.000 | 0.007 | 0.000 | 0.992 | 0.000 | 0.000 | 0.000 | 0.049527 | -4694.665527 |
| 3 | 0.014 | 0.000 | 0.733 | 0.241 | 0.012 | 0.000 | 0.001 | 0.000 | 0.000 | 0.000 | 0.686189 | -1166.876465 |
| 4 | 0.001 | 0.000 | 0.076 | 0.000 | 0.089 | 0.000 | 0.834 | 0.000 | 0.000 | 0.000 | 0.568838 | -3858.755859 |
| 5 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.001925 | -4278.995117 |
| 6 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.116 | 0.000 | 0.883 | 0.000 | 0.366590 | -6546.127930 |
| 7 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.002169 | -1089.765991 |
| 8 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.999 | 0.000 | 0.000 | 0.000 | 0.009618 | -5635.616699 |
| 9 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.999 | 0.000 | 0.000 | 0.000 | 0.009860 | -3401.843994 |
| | inlier/incorrect | | | | | | | | | | | |
| label | $p(C \mid \mathbf{X})$ | | | | | | | | | | CE | LL |
| 0 | 0.267 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.732 | 0.000 | 0.000 | 0.000 | 0.585017 | -825.450012 |
| 1 | 0.001 | 0.000 | 0.000 | 0.996 | 0.000 | 0.000 | 0.002 | 0.000 | 0.000 | 0.000 | 0.027871 | -850.614075 |
| 2 | 0.000 | 0.000 | 0.005 | 0.000 | 0.722 | 0.000 | 0.273 | 0.000 | 0.000 | 0.000 | 0.616855 | -834.262268 |
| 3 | 0.000 | 0.000 | 0.001 | 0.489 | 0.496 | 0.000 | 0.013 | 0.000 | 0.000 | 0.000 | 0.763959 | -831.182373 |
| 4 | 0.000 | 0.000 | 0.921 | 0.000 | 0.077 | 0.000 | 0.002 | 0.000 | 0.000 | 0.000 | 0.284489 | -818.538147 |
| 5 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.492 | 0.000 | 0.508 | 0.000 | 0.000 | 0.693101 | -855.898193 |
| 6 | 0.854 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.145 | 0.000 | 0.000 | 0.000 | 0.420682 | -821.650879 |
| 7 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.002 | 0.000 | 0.998 | 0.015364 | -853.953491 |
| 8 | 0.003 | 0.000 | 0.001 | 0.201 | 0.010 | 0.000 | 0.670 | 0.000 | 0.115 | 0.000 | 0.911972 | -872.540466 |
| 9 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 | 0.000525 | -847.545105 |